Remote Interactive Visualization and Analysis (RIVA) Using Parallel Supercomputers

P. Peggy Li
William H. Duquette
David W. Curkendall

Jet Propulsion Laboratory
Pasadena, CA, 91109

[illegible]

Abstract

JPL's Remote Interactive Visualization and Analysis System (RIVA) is described in detail. RIVA's kernel is a highly scalable perspective renderer tailored especially for the demands of large datasets beyond the sensible reach of workstations. The algorithmic details of this renderer are deseribed, particularly the aspects key to achieving the algorithm's overall scalability. The paper summarizes the performance achieved for machine sizes up to more than S00 nodes and for initial input image/terrain bases of up to a gigabyte.

The RIVA system integrates workstation graphics, massively parallel computing technology, and gigabit communication networks to provide a flexible interactive environment for scientific data perusal, analysis and visualization. Early experience with using RIVA to interactively explore multivariate datasets is reported and some example results given.

Keywords: 3-D perspective terrain rendering, forward mapping, scientific visualization, massively parallel processor (MPP)

1. Introduction

The goal of the Remote Interactive Visualization and Analysis System (RIVA) is to make possible the interactive exploration of the largest of NASA's scientific datasets using high speed networking and parallel supercomputing technology. To achieve this goal, it must be possible for an individual investigator to command that perhaps 1-100 gigabytes of data be brought online from his workstation and activated for fully interactive exploration, viewing, and analysis.

Accordingly, the RIVA Project was conceived to carefully construct some of the key elements needed and gain early experience with the use of large scale computing and associated high speed networking facilities. At Caltech/JPL two massively parallel processors (MPP's) are available for use - a 256 processor Cray T3D and a 512 node processor Intel Paragon. The CASA gigabit network interconnects them and provides a photonic based extension network to individual laboratories at several locations around JPL and the Campus. With these facilities, it is, for the first time, possible to remotely command highly data intensive data/image manipulation and have the results delivered instantaneously to the investigator's work station or frame buffer.

To do this, RIVA is built around a three dimensional perspective terrain renderer as a kernel and surrounded with an orchestrated set of capabilities allowing the selection of data (sets), the viewing point, and, if multiple datasets are to be rendered and viewed simultaneously, the relative translucence of each overlay surface.

The perspective renderer has been carefully designed for high performance and scalable operation. Scalable in this context means both: 1) nearly linear performance gain with increasing number of processors for a ***fixed size*** of the desired output image and 2). nearly constant performance with an increasing overall input database size. The renderer is a *forward mapping* type where input dataset points are individually mapped to the focal plane. Compared to the inverse method of casting rays from the focal plane to the surface [2,4], this approach lends itself naturally to the easy decomposition of the input dataset across the nodes of the MPP. But the rendered picture emerges spread rather randomly across these same nodes. A machine wide sort, reordering the local output pixels into focal plane order and essentially transitioning to an output frame decomposition scheme is then necessary. We have found that on balance, this hybrid decomposition scheme works very well to promote scalability and suffers only a small associated penalty of decomposition transition.

But RIVA is not just a parallel renderer exploring and testing various rendering techniques. It is, as we have already described, a tool for interactive scientific data base exploration and visualization. In this sense it is a system as shown in Figure 1. The data Navigator resides on a SGI workstation which hosts a low resolution copy of the dataset to be investigated. Using a GUI based on SGI's Open Inventor, the investigator selects the desired view he wishes; this viewpoint is transmitted to the supercomputer. As shown this is currently either the T3D or Paragon which contains a full resolution copy of the dataset. There the rendering(s) is done and the resultant full resolution image transmitted back over the CASA gigabit network and into a HiPPI frame buffer placed next to the workstation.

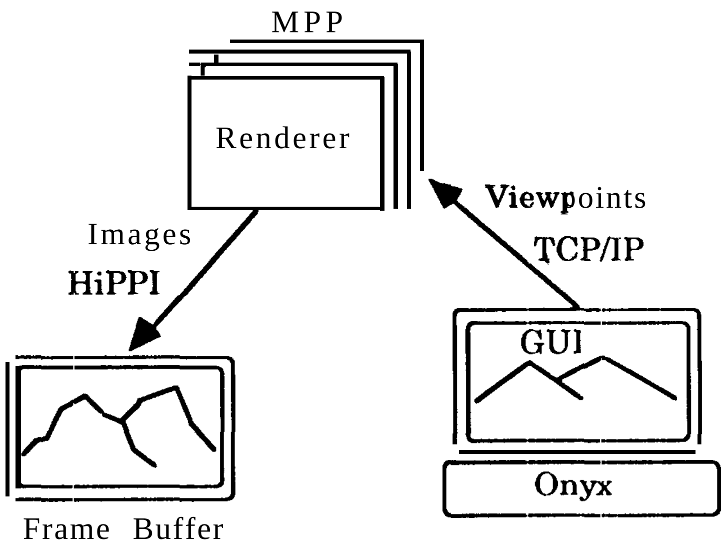


Figure 1: RIVA System Architecture

2. **Algorithm**

The kernel of the RIVA system is a parallel terrain renderer running on distributed-memory parallel supercomputers. The renderer produces 3-D perspective views of terrain using (mosaics of) remote sensing Earth or planetary images with co-registered digital elevation data. Because the underlying geometric model is that of a sphere, the renderer can accommodate global datasets; accordingly we refer to this renderer as the *whole earth renderer.*

The whole earth renderer uses a forward-mapping algorithm with object space decomposition. That is, each data input point is mapped to the output focal plane with z buffering used for hidden surface arbitration. The current implementation is an evolution of the ray identification algorithm reported earlier[1], but differs from it in many respects: 1. it supports both global datasets and regular grid 2-D surface datasets, 2. it adopts a finer grain data decomposition scheme - tile decomposition - to achieve better load balancing ,3. it performs dynamic data pyramiding for better performance, 4. it uses a sparse output image structure for more efficient memory utilization and less communication overhead, and 5. it renders multiple input datasets of different resolutions and different formats

Other than forward mapping, three different approaches have been used for terrain rendering. Representing terrain as a polygon mesh and using hardware to do polygon rendering and texture mapping is a common approach used in advanced graphic workstations. Ray casting has been used by Jet Propulsion Laboratory to produce animated fly-by movies [2], but parallelization of a my-casting renderer on a distributed memory MPP requires input data shuffling and complicated data management [4]. The data shuffling could be very costly with the size of the input data we are handling. Using shear-warp operations to achieve fast perspective terrain rendering [3,5] can not be applied to the spherical data model directly. Besides, without hardware support for fast matrix transposition, the shear-warp approach will not perform decently on a distributed memory MPP machine.

2.1 Forward Mapping

Mapping an input pixel to the output focal plane is done by first transforming the input pixel from the world coordinate to the image coordinate, then applying a 2-D scaling to the result. The image coordinate uses the camera position $\overline{O}$ as the origin, and $(\hat{u}, \hat{v}, \hat{c})$ defining the three axes. (ii, $\hat{v}$) are unit vectors defining the focal plane and $\hat{c}$ is the unit vector along the camera direction or the normal of the focal plane (Figure 2). Let $\bar{x}$ be an input pixel in the input dataset and assume all the vectors and points are represented in world coordinates. The transformation of $\bar{x}$ to the image coordinate is the dot project of vector $\overline{Ox}$ with three unit vectors $(\hat{u}, \hat{v}, \hat{c})$. The projection of $\overline{Ox}$ to the focal plane $(u,$ v) in the image coordinates can be calculated using equation (1) where f is the focal length of the camera or the distance from the camera to the center of the focal plane.

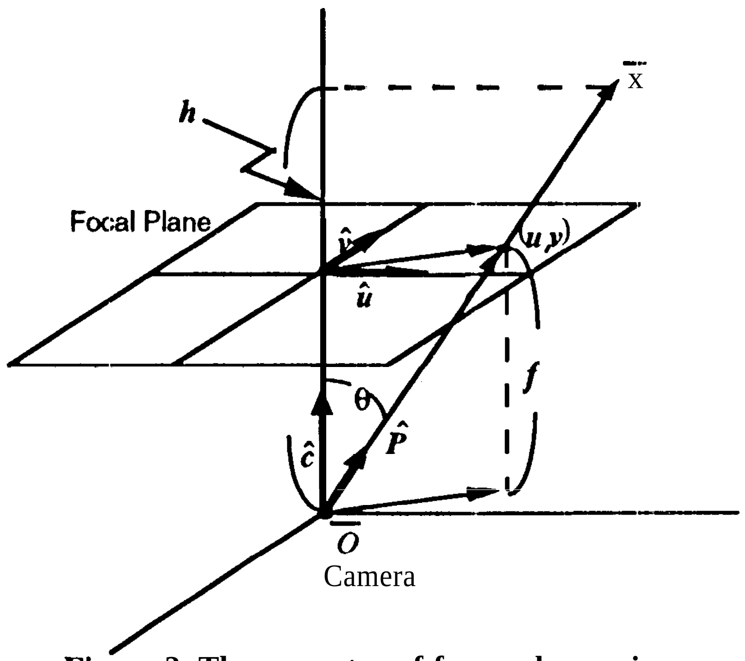


Figure 2. The geometry of forward-mapping

$$u = \frac{f}{h}\left(\hat{u} \bullet \overline{Ox}\right)$$
$$v = \frac{f}{h}\left(\hat{v} \bullet \overline{Ox}\right) \qquad (1)$$
$$\text{and } h = \hat{c} \bullet \overline{Ox}$$

2.2 Data Representation

Early in the design of the renderer it was recognized that, if very large datasets were to be supported, it would be necessary to change the underlying geometrical model away from the flat surface or ***tabletop*** model used in [1] and to a global spherical model more representative of the object being studied. The first problem encountered in doing this is the mapping of the image data to be rendered into an efficient two dimensional data structure. The two most common projections have been considered: the *cylindrical equidistant projection* and the ***Sanson-Flamsteed*** *sinusoidal projection. The* cylindrical equidistant projection maps longitude lines onto equally spaced vertical chart lines and latitude lines onto equally spaced horizontal chart lines. The features close to the polar regions are stretched, thus distorted in the map; but the mapping from a grid point on the map to its world coordinate is a straightforward scaling function.

On the globe, the relative longitude scale shifts as a function of latitude. Assuming the scale ratio for two adjacent longitude lines at the equator is 1, then the scale decreases to zero at either pole; the ratio is cos (#I, where ϕ is the latitude. By applying cos ϕ along the x axis of a cylindrical projection map, the new map maintains constant area with the globe. Because of the sinusoidal shape of the map boundary, this projection is called the Sanson-Flamsteed sinusoidal equal-area projection. Projection from a grid point on the map to its world coordinate is a simple transformation

$$\begin{aligned}\phi &= s \cdot y \\ \lambda &= s \cdot x/\cos\phi\end{aligned} \tag{2}$$

where (λ, ϕ) is the world coordinate in longitude, latitude for the data grid point (x, y), and s is the number of grid points/radian at the equator.

An image in the sinusoidal equal-area projection requires significantly less memory than does cylindrical projection. The total memory required for the former is

$$\int_{-\pi/2}^{\pi/2} \cos\phi = 2 \text{ instead of } \pi$$

yielding memory savings of $(\pi - 2)/\pi$ = 36.34%.

The whole earth renderer supports both the cylindrical and sinusoidal projections. For a large global dataset, it is more sensible to conserve memory by using the sinusoidal projection, and pay the penalty in transformation time. The cosine of latitude can be derived with a table lookup, but one extra division is required for each grid point. The sinusoidal projection does not save as much memory for datasets covering a small region of the globe; in this case it is better to use the cylindrical projection.

Internally, the renderer uses a 3-D Cartesian system for the world coordinates. The origin is at the center of the sphere, with the x and y axes lying in the equatorial plane and pointing to the prime meridian and 90”E respectively; and the z axis pointing to the North Pole. The conversion from a data grid point (x, y, z) to its world coordinate (X, Y, Z) is:

$$\begin{aligned}X &= (r + z)\cos\lambda\cos\phi \\ Y &= (r + z)\sin\lambda\cos\phi \\ Z &= (r + z)\sin\phi\end{aligned} \tag{3}$$

where z is the altitude, r is the radius of the reference. sphere and (λ, ϕ) can be derived from (x, y) using equation (2).

2.3 Data Decomposition

Input space decomposition is ideal for forward mapping. The input pixels can be projected into the image space in parallel in any order. If the input dataset is decomposed into the distributed memory of the parallel processors, the pixel projection, rasterization, and local hidden surface arbitration can be done locally and concurrently in each processor without any data exchange. Communication is only required in the transitioning to an image space data decomposition in order to merge the partial images and perform the final hidden surface arbitration. Since the input space we are dealing with is many orders of magnitude bigger than the image space, the communication cost for shuffling the partial image is much less than that of shuffling the input pixels.

Our algorithm uses static input space decomposition, In order to achieve good load balancing, an interleaved tile decomposition with a small tile size is used. The tile decomposition of a global sinusoidal dataset is depicted in Figure 3. The tiles are aligned horizontally, but not vertically. Each tile overlaps its boundaries with its neighboring tiles to avoid exchanging boundary information needed for the detailed painting of the output raster. The tiles are dealt to each processor cyclically, guaranteeing a random distribution of the tiles over the parallel processors, and thus a good load balance for any arbitrary viewpoint. The choice of the tile size is a tradeoff of memory/computation overhead vs. load balancing. For a square tile of nxn pixels, it requires $(2n-1)$ extra pixels to store the boundary data and an extra $(2n-1)/(n \cdot n) \approx 2/n$ computation overhead. When n is 64, a tile size we have experimentally determined to be near optimum, there is a 3% memory and computation overhead. It is noted that smaller tiles also imply higher computation cost in both filtering and pyramiding, operations discussed in the following sections.

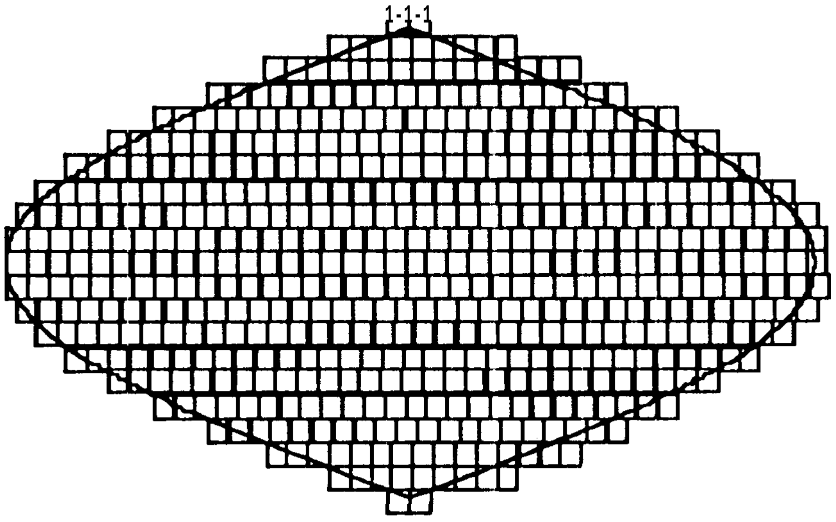


Figure 3. Interleaved tile decomposition of the input dataset

2.4 **Filtering**

Filtering is the first acceleration step in the whole earth renderer. The purpose of filtering is to efficiently eliminate the pixels in the input space that can be determined to fall outside the field of view of a given viewpoint. In the previous approach [1], a footprint area was formed by analytically intersecting rays from the viewpoint through the corners of the focal plane to the surface of zero elevation. Then a rectangular bounding box was determined using that footprint and the projection of the viewpoint in the terrain surface. Anything outside the bounding box was excluded in the field of view of this particular viewpoint. This footprint approach, while straightforward for tabletop renderers, is much more difficult when using a spherical data model. For example, when rendering a distant view that encompasses a whole hemisphere, none of the corner rays intersects the dataset at all! Moreover, because of the quadratic nature of the underlying surface, the general ray intersects twice, once on the front side and once on the back surface. These and other difficulties led us to adopt quite a different approach to filtering for the whole earth renderer.

The approach used is to inspect the location of each tile to determine if it is possible for any of its pixels to be needed for the final image. Two tests have been devised to make this determination. The *horizon filter* determines whether a tile is out of sight over the horizon, and the *tile filter* determines whether any part of the tile is included in the focal plane. Both filters are conservative: including a tile in the detailed computations when none of its pixels are in the final image is more acceptable than excluding a tile that was needed.

Horizon Filter

In Figure 4, a viewpoint O is distance h above the zero elevation surface of the sphere. The furthest point it is possible to see from this viewpoint is one at the maximum elevation a along the line of sight tangent to the zero (or minimum) elevation surface P. Q is such a point.

$$\overline{OQ} = \overline{OP} + \overline{PQ} = \sqrt{h^2 + 2hr} + \sqrt{a^2 + 2ar} \qquad (4)$$

where r is the radius of the zero-elevation sphere, a is the maximal elevation of the data above the sphere and h is the distance between the viewpoint and the zero elevation surface. For each tile, the distance is computed from the viewpoint to the central pixel of the tile with an elevation the average of the highest and the lowest pixels in the tile. If the distance is greater than $\overline{OQ}$, the tile is excluded from the field of view. Even in the limiting case of large h, this filter will eliminate almost half of the tiles. For a more interesting close-in view, even the largest datasets are brought to very manageable data sizes with but little computational overhead.

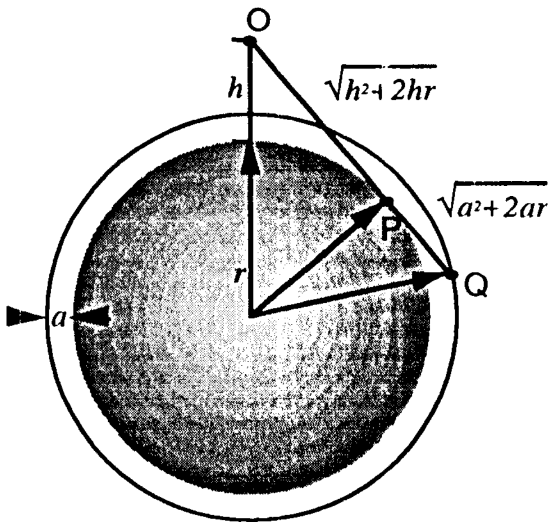


Figure 4. The horizon geometry for a lumpy sphere

Tile Filter

The tile filter is applied to the tiles surviving the horizon test just above and is more in the spirit of the original footprint filter used in [1]. First, the four corners of a tile are projected to the image space. If any of the corners fall inside the focal plane, the tile is admitted. If this test fails, one final computation is performed before final elimination is determined. This final check visualizes the tile as a rectangular volume which encloses not only the latitude/longitude excursions of the tile boundaries, but bounds this volume by the maximum and minimum height elevations for that particular tile. This final check maps the center point of this volume to the focal plane and calculates first order effects of excursions in latitude, longitude and elevation. This helps guard against the possibility that all four corners miss but some interior regions are really in the picture.

In Figure 2, let $\bar{x}$ be the central pixel of a given tile with an elevation the average of the highest and the lowest points within that tile. The cosine of the angle between $\overline{Ox}$ and the view direction $\hat{c}$ can be calculated by the dot product of the unit vector}= $\overline{Ox}/\|\overline{Ox}\|$ and $\hat{c}$. In equation (5), we add to the basic dot product three increments that represent the linear change in the dot product in going from the average altitude to either the highest or the lowest point, and the gradient change across the latitude and the longitude width of the tile. This incremented sum is then compared with the cosine of the field of view to determine acceptance.

$$\hat{P}\cdot\hat{c}+\left|\frac{\partial \hat{P}\cdot\hat{c}}{\partial a}\right|\left(\frac{a_{\max}-a_{\min}}{2}\right)+\left|\frac{\partial \hat{P}\cdot\hat{c}}{\partial \lambda}\right|\frac{\Delta\lambda}{2}+\left|\frac{\partial \hat{P}\cdot\hat{c}}{\partial \phi}\right|\frac{\Delta\phi}{2} \qquad (5)$$

where $a_{\max}$ and $a_{\min}$ are the highest and the lowest elevation of the tile, and $\Delta\phi/2, \Delta\lambda/2$ are the semi latitude and longitude widths of the tile. The three partial derivatives can be derived using equations (6) to (8)

$$\frac{\partial \hat{P}\cdot\hat{c}}{\partial a}=\hat{P}\cdot\frac{\partial \hat{c}}{\partial a}=\hat{P}\cdot\left(\frac{1}{\|\overline{Ox}\|}(\hat{x}-(\hat{x}\cdot\hat{c})\hat{c})\right) \qquad (6)$$

$$\frac{\partial \hat{P}\cdot\hat{c}}{\partial \lambda}=\hat{P}\cdot\frac{\partial \hat{c}}{\partial \lambda}=\hat{P}\cdot\left(\frac{1}{\|\overline{Ox}\|}\left(\frac{\partial \bar{x}}{\partial \lambda}-\left(\frac{\partial \bar{x}}{\partial \lambda}\cdot\hat{c}\right)\hat{c}\right)\right) \qquad (7)$$

$$\frac{\partial \hat{P}\cdot\hat{c}}{\partial \phi}=\hat{P}\cdot\frac{\partial \hat{c}}{\partial \phi}=\hat{P}\cdot\left(\frac{1}{\|\overline{Ox}\|}\left(\frac{\partial \bar{x}}{\partial \phi}-\left(\frac{\partial \bar{x}}{\partial \phi}\cdot\hat{c}\right)\hat{c}\right)\right) \qquad (8)$$

where $\hat{x}$ is the unit vector from the origin of the world coordinate to the central pixel $\bar{x}$.

Although the tile filter involves complicated computations and has to be repeated for every tile, the result is potentially more efficient than the original footprint filter. The resulting bounding area covered by the accepted tiles need not necessarily be rectangular and could conform better to the actual footprint.

The horizon filter and the tile filter can be computed concurrently in all the parallel processors. The compute time is proportional to the number of tiles in each processor, thus is linearly scaled down with number of processors. In our test case, for a tile size of 64x64 pixels, the filtering time is less than 1% of the total rendering time.

2.5 Data Pyramiding

Data pyramiding is the second acceleration step used in the whole earth renderer. Data pyramiding is an image scaling technique that has been used in terrain rendering to reduce computation and eliminate aliasing problems [2] [3]. For a large input dataset with hundreds or even thousands of mega pixels, it is important to be able to view the data at different levels of detail, from a perspective that overlooks the entire dataset down to a small area for a close look of the terrain. From an overlook perspective, the scale from the input space to the image space could be hundreds to 1, in which case there is no need to render the input data at its highest resolution. An input data pyramid is built by repeatedly dividing the input image by a factor of 2 in each dimension. The input pyramid may be pre-generated and stored in memory. A multi-level input pyramid requires about 1/3 more memory than the single highest resolution image, which could be significant when trying to fit large datasets to any particular machine. Alternatively, the program can generate the input pyramid on the fly. The whole earth renderer currently uses the latter approach.

In the filtering step, each processor constructs a list of tiles that were accepted for this particular viewpoint. A pyramid level is then computed for each tile in the list. A pyramid level of p implies that the tile will be divided by 2^{p-1} in each dimension; thus, a p of 1 implies no pyramiding; of 2, that the tile will be divided in half in each dimension; of 3, that the tile will be divided in fourths in each dimension, and so on.

The pyramid level is computed as follows: 1. project the four corner pixels of a given tile to the focal plane to form a footprint of the tile, 2. calculate the area of the bounding rectangular of the footprint and find the ratio of the area to the size of the tile, 3. use the square root of the ratio to determine the pyramid level for this tile. Once the pyramid level is determined, the left upper corner pixel of a n x n pixel-square (where $n = 2^{P-1}$) will be forward-mapped to the focal plane and painted the average color of the n x n pixels in this square. The neighboring tiles maybe rendered at different pyramid levels while still preserving continuity in the image space because the corner pixel of a pixel-square is used to do the forward mapping and the boundary row and column of two adjacent tiles arc available to both tiles.

Smaller tiles yield better load balancing, which can improve the speed significantly for close-up views; however, the number of pyramid levels is bounded by the size of the tile. For a tile of 64x64 pixels, the maximal pyramid level is $\log_2 64 = 6$. Therefore, it may take longer to render a distant perspective if the tile size is smaller. Smaller tiles also increase storage and computation overhead for the duplicated pixels at the tile boundary, and increase the amount of filtering. In our experience, tiles of 64x64 pixels arc a good balance.

2.6 Image Compositing

In our previous approach [1], a binary-tree merging algorithm is used to create the final image. The binary-tree merging algorithm is a parallel merging algorithm that takes $\log_2 n$ steps to swap the partial images for n processors. The final image is evenly distributed over the parallel processors with each processor holding l/n th of the total lines of the final image. In the original approach, each processor allocates a memory space for the entire output image. The local node renders into this memory space, painting pixels into what will be their final locations. This organization, while convenient and while it produced good results quickly, was fatal in two regards.

First, although the partial output image in each node gets sparser as the number of processors increases, the memory used and the total message size from each processor remains constant. Double the number of processors and the total communication time for image merger gets slightly longer. Thus as the number of processors increases and computation time diminishes, communications will quickly come to dominate the overall rendering time and prohibit the desired scalability. With current commercial machines such as the Paragon or T3D, only about 32 nodes could be effectively employed before communications sensibly blocked further speed up. Second, since each node needs to reserve a block of memory equal to the resultant picture size, very high fidelity frames become impossible due to lack of per node memory regardless of the number of processors employed.

In the whole earth renderer, a distributed output data structure is designed to store the sparse output image being generated in each node. The new data structure has three features: 1. only the output pixels being painted in the local processor are stored, 2. random access to this data structure is optimized, and 3. the memory management cost is minimized. The seeond feature is important because the forward mapping is done in the order of input tiles; therefore, there is no predictable access order to the output image. Moreover, one output pixel may be projected by multiple input pixels, and thus be accessed more than once.

The data structure for the output image is an array of (raster) line pointers. The length of the array is the number of lines of the final image. Each pointer in the array points to a sorted doubly-linked list of pixel blocks representing one output line. The pointer is null if no pixel in this line is painted in the current processor. Each entry of the doubly-linked list has a header and a block of contiguous output pixels. The header contains the first x position of the pixel block, and the blocks arc sorted by this x value. Each output pixel in the pixel block is a four-tuple (r, g, *b, z)* where (r, g, *b*) is the color of this pixel and z is the distance from the viewpoint to the closest input pixel that is mapped to this output pixel. The block size is fixed and is 20 or 24 in our current implementations. The header of each linked list has three pointers, one pointing to the first block, the second to the last block and the last to the block that was last accessed. Figure 5 depicts the data structure for the sparse output image.

When an output pixel is painted, the linked list for its line is traversed starting from the last accessed block. If no block contains this pixel, a new pixel block is allocated and inserted into the linked list. If the block exists but the pixel has not yet been painted, the color and the z value of the input pixel is assigned to this output pixel. If the pixel has been painted, the stored z value is compared with the new z value. The pixel is repainted if the new z value is smaller than the stored one; the new pixel's color is blended into the output pixel if the two z values are close to each other; the new pixel is discarded if the new z value is larger than the stored one.

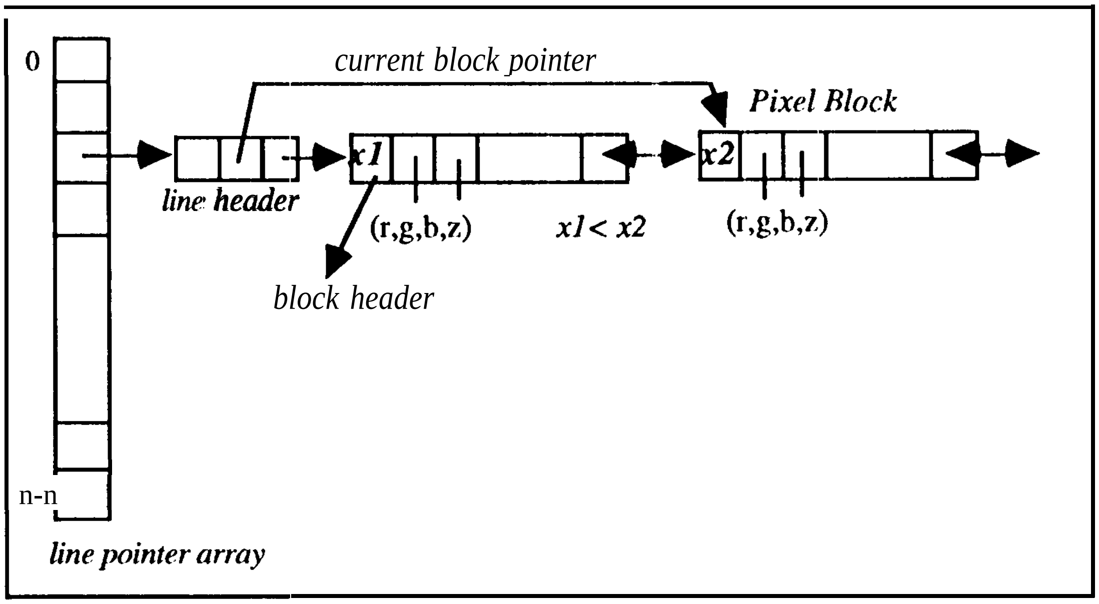


Figure 5. The data structure for the sparse output image

The binary-tree merge algorithm described in [1] is used in the whole earth renderer as well, with a few changes. In [1], the data to be swapped from processor to processor was in contiguous memory. This is clearly not the case with a linked-list implementation. Early versions of the whole earth renderer packed the pixel blocks to be swapped into a buffer; after swapping, the blocks were unpacked into a linked list, and the partial images were merged. The pixel blocks were allocated from the heap, using *malloc()*. Significant time overhead was associated with the packing/unpacking, and with block allocation/de-allocation. The current implementation greatly reduces this overhead.

After all input data has been read, a single large block of memory is allocated from the heap to stem the sparse output structure. Further allocation from this block is handled by the renderer itself. A memory map of the block is shown in Figure 6. At the bottom of the block is the line pointer array. The top of the block is reserved as a communications transfer area. Pixel blocks are allocated in between, just after of the line pointer array. A pixel block allocation involves nothing more than incrementing the block pointer. All pixel blocks can be de-allocated by resetting the block pointer. Thus, allocation/de-allocation overhead is minimized.

At each step of the binary tree merge, half of the image lines (and thus, on average, half of the pixel blocks) on each processor must be swapped with another processor. The blocks to be transferred must be packed into a contiguous block of memory. This is accomplished by sorting the pixel blocks in the block area. After the sort, the blocks for the lines which remain on the processor are at the bottom of the pixel block area; the blocks to be swapped are above them. Since linked lists link the blocks in sorted order, they can be sorted into contiguous memory in linear time. This is no slower than packing into a buffer, and requires no additional memory.

When the sets of pixel blocks arc to be swapped, each processors writes its set into the transfer area on its neighbor, along with a vector containing the number of pixel blocks on each image line. With this information, the receiving processor can merge the blocks into its own data structure without any unpacking. Thus, the packing/unpacking time is minimized.

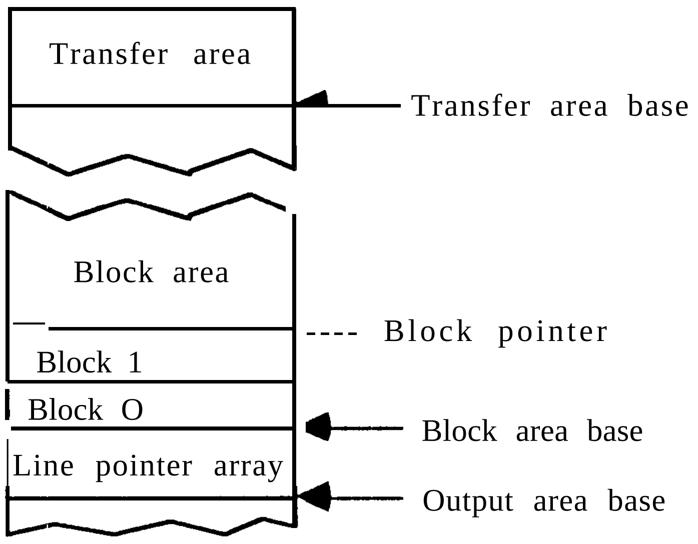


Figure 6: Output Area Memory Map

2.7 Multiple Dataset Rendering

The cut-rent version of the whole earth renderer is capable of loading multiple datasets at different resolutions, rendering them, and compositing the output. For example, the renderer can overlay a global geology map and high-resolution terrain insets on top of a global mosaic of the planet Mars. Each dataset consists of an RGB image of some planetary region in cylindrical or sinusoidal projection, with a single underlying digital elevation model.

Input decomposition is as described above. Each dataset is tiled and distributed cyclically to the parallel processors. The input datasets are processed sequentially on each processor, and the final image is composite in parallel.

As with the single dataset case, the sparse output structure is an array of output image lines, each of which is represented as a linked list of pixel blocks. Each output pixel in the pixel block is an m x 4-tuple, containing one (r,g,b,z) four-tuple for each of the m input datasets. Thus, all datasets arc rendered into a single sparse output structure, but a separate output image is rendered for each. The sparse output structure is merged using the same binary-tree merge algorithm used in the simple case.

After merging, the m images must bc combined or cornposited into a single output image. This could be done in several ways, depending on the specific application. For example, it is possible to do Z-buffering for hidden surface arbitration between the datasets since the Z-buffer values have been preserved for each image.

The current implementation assumes that the datasets are to be overlaid one on top of the previous. Therefore, the final output image can be compositcd in a back-to-front order using an OVER operator as in (9):

$$\begin{aligned} c_o &= (1-p_n)\cdot c_i + p_n \cdot c_n \\ p_o &= (1-p_n)\cdot p_i + p_n \end{aligned} \tag{9}$$

where c denotes the color, p denotes the opacity, o denotes the combined output, i denotes what is already combined and n is the new point. The opacity of a given dataset can be adjusted interactively from the GUI. If opacity of the top dataset is 1.0, the top most image is completely opaque; the images in the back only show through in those areas where the top image has no pixels. This compositing technique is based on the assumption that the multiple imaging datasets are mapped to the same digital terrain.

3. RIVA System Overview

The whole earth renderer can be used in batch mode, but its primary use is as one of the two main software components of the Remote Interactive Visualization and Analysis (RIVA) system. As depicted in Figure 1, RIVA allows a user to explore a large planetary dataset (or datasets) interactively, flying around the dataset using a graphical user interface (GUI). The GUI is the second of the two main software components. The whole earth renderer runs on a massively parallel machine, such as a Cray T3D or an Intel Paragon. The GUI, called the SpaceFlyer, runs on a Silicon Graphics workstation with texture map hardware, such as an SGI Onyx.

The SpaceFlyer, which is based on SGI's OpenInventor system, is used to navigate around a very low-resolution copy of the planetary dataset. Viewpoints are continually sent to the renderer over a standard Internet TCP/IP connection. The renderer produces images of the terrain as seen from the viewpoints, and sends them to a HiPPI (High Performance Parallel Interface) frame buffer. The frame buffer is typically adjacent to the user's workstation. As explained in the Introduction, the CASA gigabit photonic network is the medium for the long haul communication; the re conversion to HiPPI protocol is done to be compatible with commercial workstation and framebuffer interfaces.

In addition to viewpoints, the Spaceflyer can send a variety of commands to the renderer. The user can vary the terrain's vertical exaggeration, and control the display of multiple datasets. Rendering of individual datasets can be enabled and disabled; this allows the user to increase the frame rate by disabling any loaded datasets not of immediate interest. Composition parameters (e.g., opacity) can also be set.

The SpaceFlyer also provides a simple animation capability. While flying about the dataset, the user can save and edit a sequence of flight path control points. Upon user request, the SpaceFlyer splines a complete flight path (30 viewpoints per second of animation) along the sequence, using cubic spline interpolation. The flight path can be previewed in the SpaceFlyer, and saved to a file for later rendering in batch mode. During the preview, the viewpoints are sent to the renderer, which will render and display approximately every 15th viewpoint. (Preview takes place at the maximum speed of the workstation, and is generally less than 30 viewpoints per second.) While simple, this has proved both effective and convenient: animation planning can be conducted at the full resolution of the renderer.

4. **Timing Results**

4.1 Renderer Implementation

The whole earth renderer works on both the Intel Paragon supercomputers and the Cray T3D Massively Parallel Processors. The Intel Paragon at the Caltech Concurrent Supercomputing Consortium (CCSC) has512 computing nodes with 32 Mbytes memory per processor. Each i860 processor in the Paragon has a peak performance of 75 Mflops for 64-bit arithmetic. The nodes are connected in a two-dimensional mesh with 200 megabytes per second hi-directional bandwidth. The compute nodes provide an aggregate peak speed of 38.4 gigaflops and a total of 16.4 gigabytes of memory. The Cray T3D MPP at JPL has 256 Processing Elements (PEs). Each PE in the Cray T3D consists of a DEC Alpha microprocessor with a peak performance of 150 Mflops and a 64 Mbytes DRAM memory. The PEs are connected by a bi-directional 3-D torus system interconnect network with 300 megabytes per second bi-directional bandwidth. The aggregate peak performance and total memory capacity of the JPL 256 PE Cray T3D is exactly the same as the CCSC 512 node Intel Paragon.

The renderer is written completely in C. For inter-processor communication it uses the NX library on the Intel Paragon and the shared memory library (*shmem_put*, *shmem_get*) on the Cray T3D. The renderer runs either as a batch program or interactively with the Spaceflyer GUI. A gateway program is used to provide the interface between the renderer and the network or other I/O devices. On the T3D the gateway program runs on its front-end YMP processor; while on the Paragon, the gateway runs on a separate compute node. The existence of the gateway program allows the renderer to be independent of the network interface and also introduces pipelining parallelism between the rendering process and the I/O processing.

4.2 **Datasets**

Two remote imaging datasets were used for the timing runs: the Southern California desert and the planet Mars. In addition, the T3D code was also run with a global Geology map and a high-resolution inset of the Ares Vallis and Tiu Vallis regions on top of the Mars dataset. All datasets consist of an RGB image and a co-registered 16-bit digital elevation model.

Southern California Desert

This dataset covers a region stretching from San Bernardino north to central Nevada at a resolution of 30 meters. The RGB image was produced by the Landsat Thematic Mapper, and includes three of the seven Thematic Mapper bands. The digital elevation was provided by the U.S. Geological Survey. The dataset is 17,420 rows by 7,435 columns, or 31,941 65x65-pixel tiles. The total size of the dataset is about 645 megabytes. Figure 10 is a high-resolution perspective view of Death Valley with a vertical exaggeration of 2.

Mars Global Mosaic

This dataset covers the entire surface of the planet Mars at a resolution of one kilometer. The images making up the mosaic came from the Viking spacecraft which orbited Mars in the late 1970's. The mosaic and accompanying digital elevation model were assembled by the U.S. Geological Survey. The dataset is 23,040 rows by 11,520, or 41,715 65x65-pixel tiles. Because the data is in a sinusoidal projection, not all of the pixels contain data; empty tiles arc not included. The total size of the dataset is thus about 842 megabytes. Figure 11 is a high-resolution perspective view of Valles Marineris of Mars with a vertical exaggeration of 5.

Mars Geology Database

This dataset covers the entire surface of the planet Mars at a resolution of four kilometers. It is a digital map of the surface composition of Mars, assembled by geologists at the U.S. Geological Survey. The original map was grayscale; the map was converted to RGB using an arbitrary RGB palette. Also, a digital elevation model at four kilometers was averaged down from the DEM used with the Mars Global "Mosaic. The dataset is 5760 rows by 2880 columns, and is also in a sinusoidal projection, yielding 41,685 17x17-pixel tiles. The total size of the dataset is about 60 megabytes. Figures 12 and 13 show a global view of Mars before and after overlaying the Mars geology map. The overlay opacity is 2.0.

Mars Ares-Tiu Mosaic

This dataset covers the Ares Vallis and Tiu Vallis regions of Mars at a resolution of 30 meters. It was assembled by scientists at the Jet Propulsion Laboratory, and is centered on the proposed landing site for JPL's Mars Pathfinder mission. The dataset is 6144 rows by 9750 columns, or 14,688 65x65 pixel tiles. The total size of the dataset is about 237 megabytes. Figure 14 is a close view of Ares Vallis region. The 30-meter data patch was originally in gray scale, and was colorized with the color of the Global Mars mosaic. The color of the two datasets is not perfectly matched and the boundary of the high-resolution dataset can be easily seen. In this picture, the opacity of the high-resolution dataset is set to 1.0.

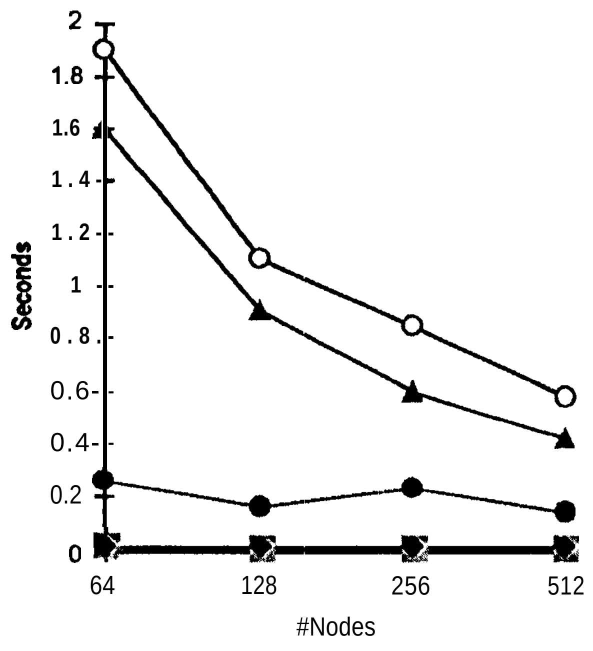
4.3 Timing Results

Timing tests were run on the JPL 256-PE Cray T3D, and on the Caltech 512-node Intel Paragon. Figure 7 shows the timing results for the Mars and Southern California Desert datasets, on each machine, using different numbers of processors. Each chart shows the average number of seconds/frame total and for several subphases of the algorithm. Filtering time is the time spent in calculating horizon filter and grid filter for all the tiles. Computing time is the time to project the input pixels into the focal plane, rasterize them and perform z-buffer hidden surface removal. Merging time is the time to merge the partial output images using the binary-tree merge and combining time is, to collect the distributed final image and send it out to the gateway processor. In both' machines, most of the time is spent projecting and rasterization; filtering the input tiles and producing the final frame are consistently small. Filtering speeds up linearly as processors are added; combination, being a gather operation, slows down slightly. The binary-tree merge step scales linearly on the T3D and is small, but is significant on the Paragon, due to the higher intra-node communications time.

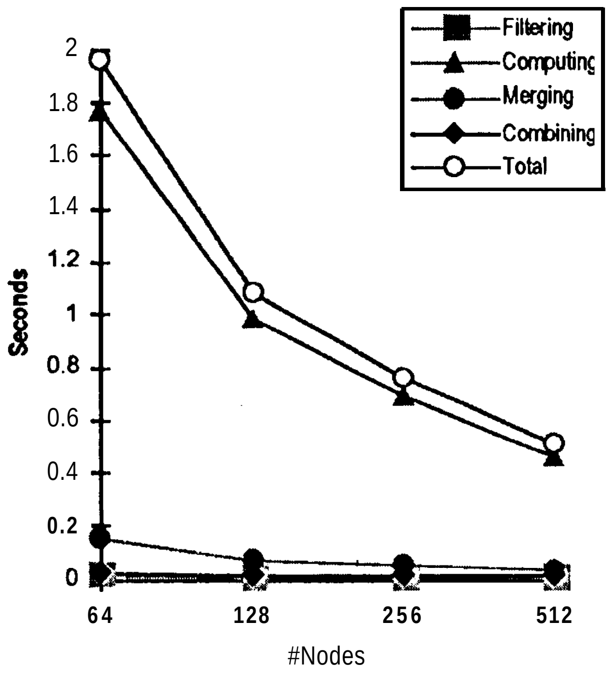
Figure 8 compares the T3D and Paragon results directly. For this application, we consistently get about twice the performance from the T3D for the same number of processors; given the relative performance of the Alpha and i860 processors, this is quite reasonable. Allowing for the difference in processor speeds, the algorithm performs comparably on both machines. The program scales well first with about 80% efficiency, but decreasing to 40% at their largest configuration for both machines. The major cause ,,, for decreasing efficiency is that the load on each processor gets more unbalanced when fewer tilers are computed locally with increasing number of processors. On the T3D, the program speeds up about four times when number of PEs increases from 32 to 256. On the Paragon, we also see about four times speed up by increasing number of nodes from 64to512. The total frame rate is about 2.5 frames/second on 256 PE T3D and 2 frames/second on 512 node Paragon.

Figure 9 charts seconds/frame vs. frame size on the T3D using 128 processors. The frame sizes quadruple from left to right, so the increase is mom nearly linear than it appears. The binary-tree merging and image combining steps increase slightly with the larger frame sizes, but the largest increase is due to painting the output pixels.

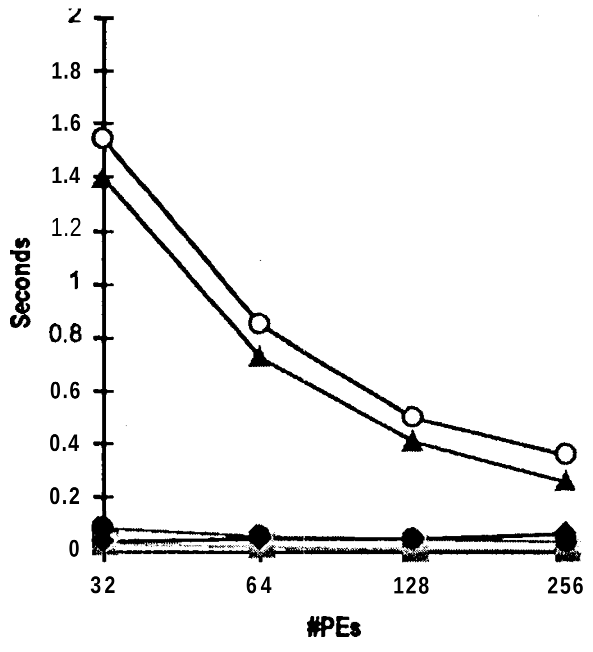
We also compared the timing difference between single-dataset rendering and multiple-dataset rendering. With a fixed output frame size, the rendering time increases with the total size of the input datasets, which implies that the time for compositing the final image from the multiple datasets is negligible. The single dataset wc used was the Mars Global Mosaic; the multiple dataset case was Mars global mosaic added with the Mars geology map and the Ares Vallis/Tiu Vallis high-resolution inset. The rendering time for the three datasets is consistently about 50% longer than for the single dataset; as the three datasets contain 1.5 times as many tiles.



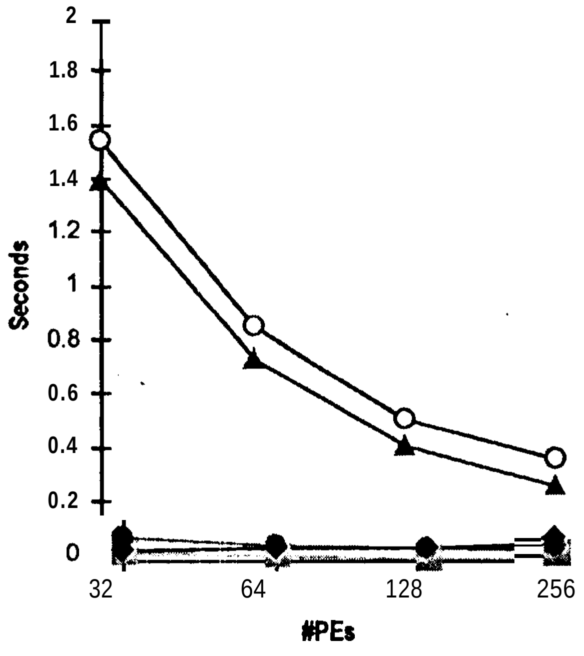
(6a)



(6b)



(6c)



(6 d)

Figure 7. The render timing vs. the number of processors. (a) on the Intel Paragon using the southern California desert dataset, (b) on the Intel Paragon using the Mars global mosaic, (c) on the Cray T3D using the southern California desert dataset, and (d) on the Cray T3D using the Mars global mosaic.

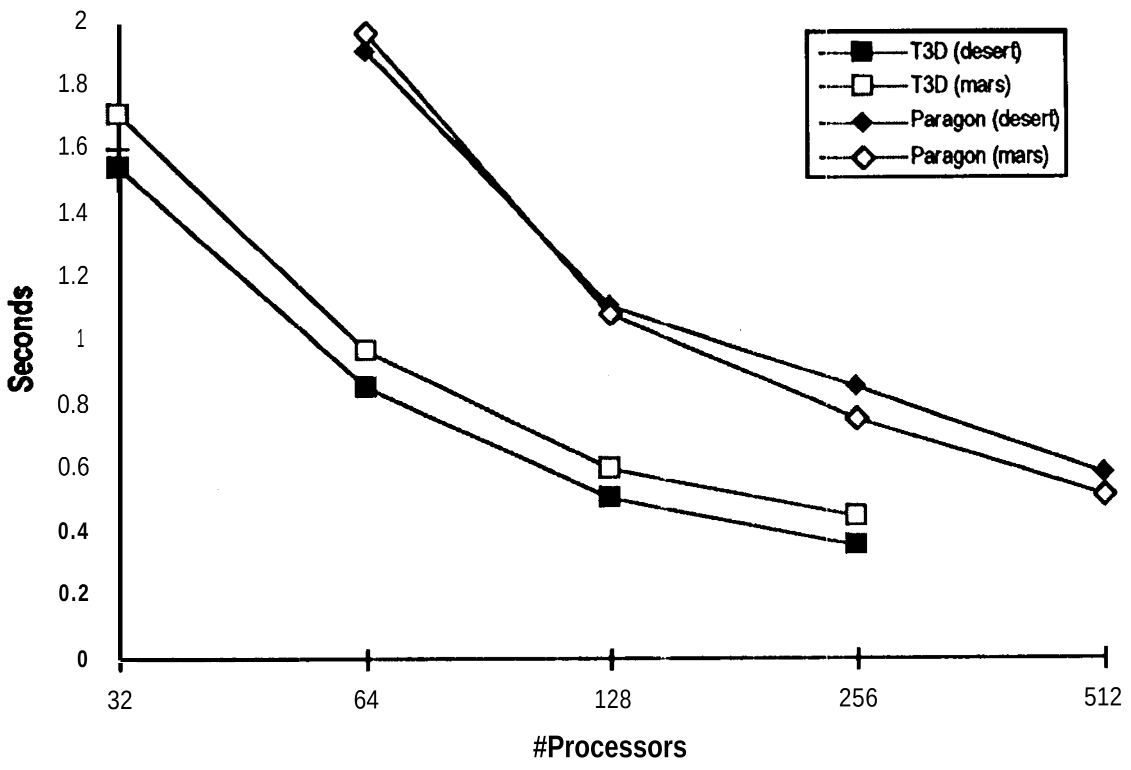


Figure 8. The timing companion between the Cray T3D and the Intel Paragon

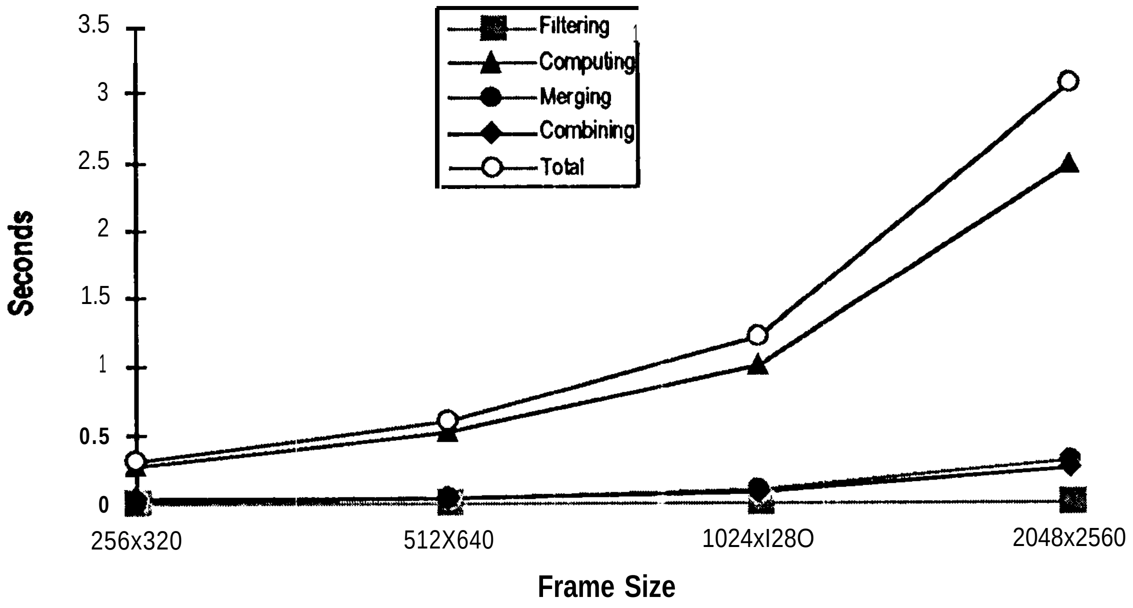


Figure 9. The render timing vs. the size of the output frames

The research described in this publication was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

5. Conclusions

The RIVA system described in this paper enables the interactive exploration of very large earth and planetary datasets which are far beyond the memory and the CPU capacities of the fastest workstations. The RIVA system allows investigators to explore their datasets from different perspectives, at different levels of detail, and even using multi-phenomenologies. The whole earth renderer was designed to be scalable to very large datasets and very large MPP machines. The interleaving tile decomposition results in a reasonably good load balancing and the sparse output image structure minimizes the inter-processor communication overhead. The filtering and pyramiding techniques reduce the amount of computation to be more or less independent of the total size of the input data.

Although the current frame rate of the whole earth renderer is fast enough to meet our interactive exploration goals, additional speed-up is always desirable. As can be seen by the timing figures, the bottleneck of the current implementation is in computation: input pixel projection, rasterization and z-buffer arbitration. The rasterization and z-buffer arbitration time is almost three times longer than that of projection. More detailed inspection of the current rasterization mechanism is needed for more speedup. The load balancing becomes poor when the viewpoint is very close to the terrain and the focal plane is painted by only a few tiles. A dynamic load balancing scheme that subdivides the input tiles and redistributes them is under investigation.

But the primary thrust of R] VA-development now is towards making the system more useful to the scientist. To this end wc are undertaking the following treks: 1. we are extending the renderer to support additional kinds of planetary data, such as the transformations and combinations of all seven bands of Landsat TM data, 2. we are extending the GUI and GUI-renderer interface to give the user additional control over the renderer and to provide additional information about the data to the user, and 3. we are examining ways of delivering the rendered images across slower network connections.

6. Acknowledgments

This work was sponsored in part by the NASA High Performance Computing and Communications (HPCC) Program Earth and Space Science (ESS) Project and the NASA Information Systems Office. Access to the Intel Paragon and the Cray T3D computers were provided by the California Institute of Technology and Jet Propulsion Laboratory. We would like to acknowledge the contributions of Dan Katz, Cray Research, Inc., for his many refinements and optimization of the T3D code. Bruce Greer and Mahesh Rajan of Intel also provided a review of the implementation. Terry Cole and Ron Blom have been constant in their encouragement and have provided many helpful discussions. The RIVA software is available via the HPCC Software Exchange.

References

[1] P. P. Li and D.W. Curkendall, *Parallel Three Dimensional Perspective Rendering,* Proceedings of the Second European Workshop on Parallel Computing, March 1992, pp 320-331.

[2] D. Stanfill, *Using Image Pyramids for the Visualization of Large Terrain Data Sets,* International Journal of Imaging Systems and Technology, Vol.3, 1991.

[3] J. Kaba and J. Peters, *A Pyramid-based Approach to Interactive Terrain Visualization*, Proceedings of 1993 Parallel Rendering Symposium, pp 67-70.

[4] S. Groom, S. Watson, and E. DeJong, *Using the Delta for Solar System Visualization,* Information Systems Newsletter, NASA Office of Space Science and Applications, Issue 26, October 1992.

[5] P. Robertson, *Fast Perspective Views of Images Using One-Dimensional Operations* IEEE Computer Graphics & Applications, February 1987, pp 47-56.

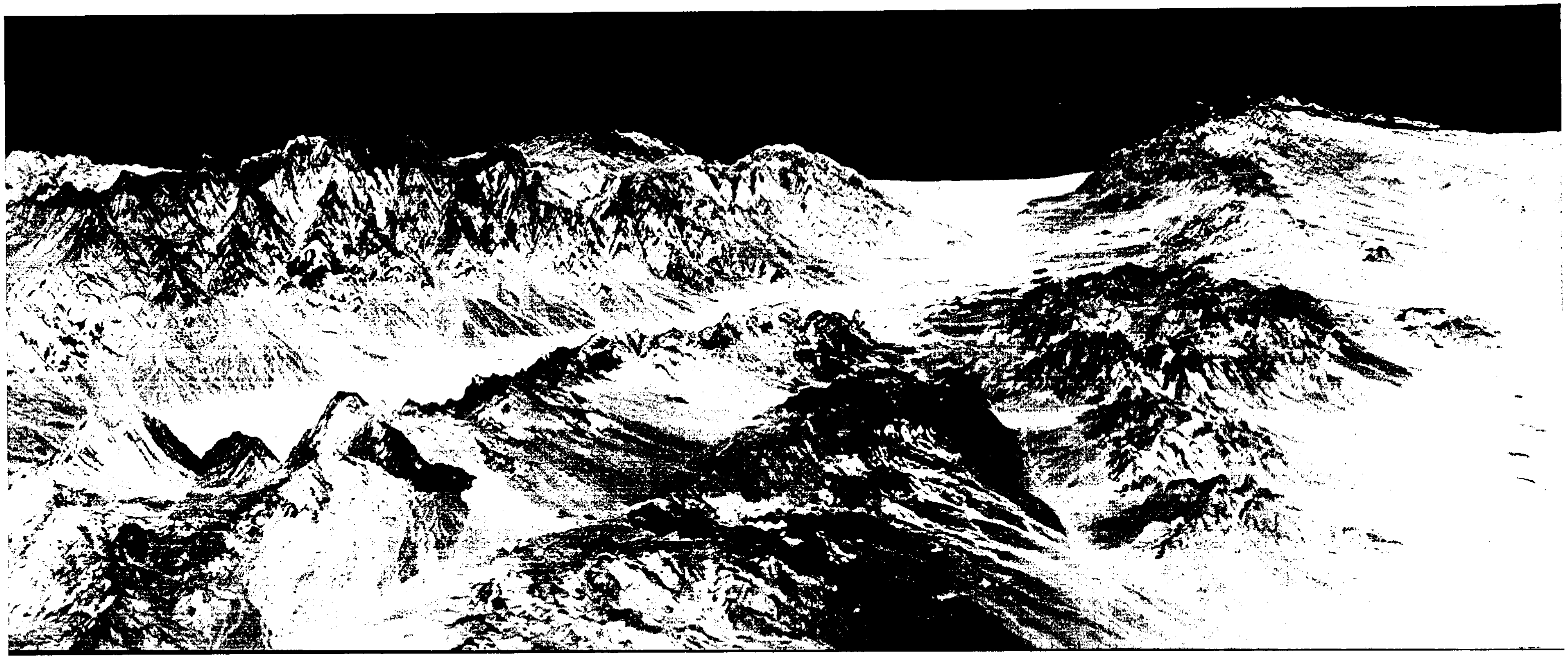


Figure 10. **A** 3-D Perspective **View of** Death **Valley** rendered from **the** Southern California Desert Landsat dataset

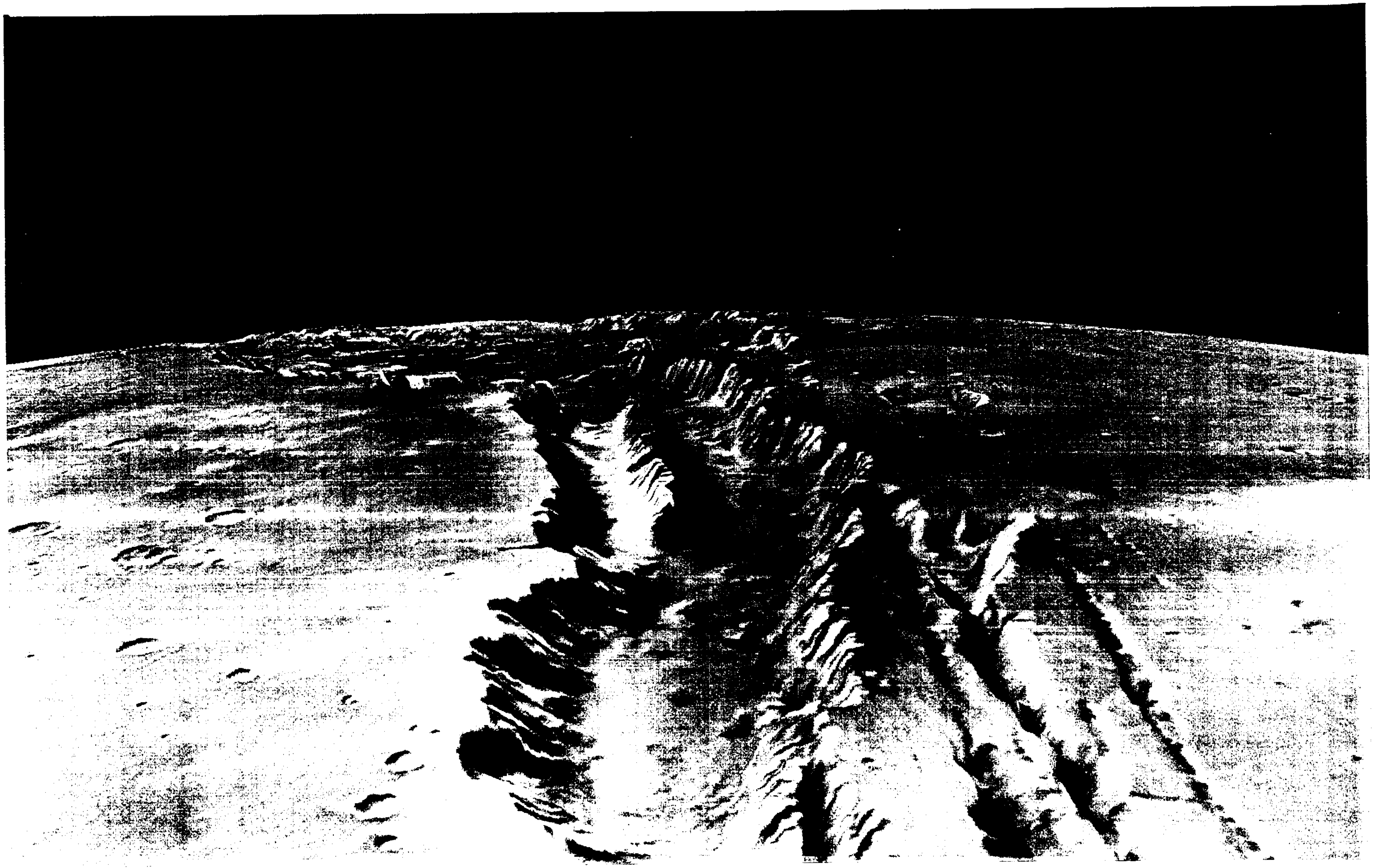


Figure 11. **A** 3-D Perspective **View of Vanes Marineris of** Mars rendered from the **USGS Mars global color mosaic.**

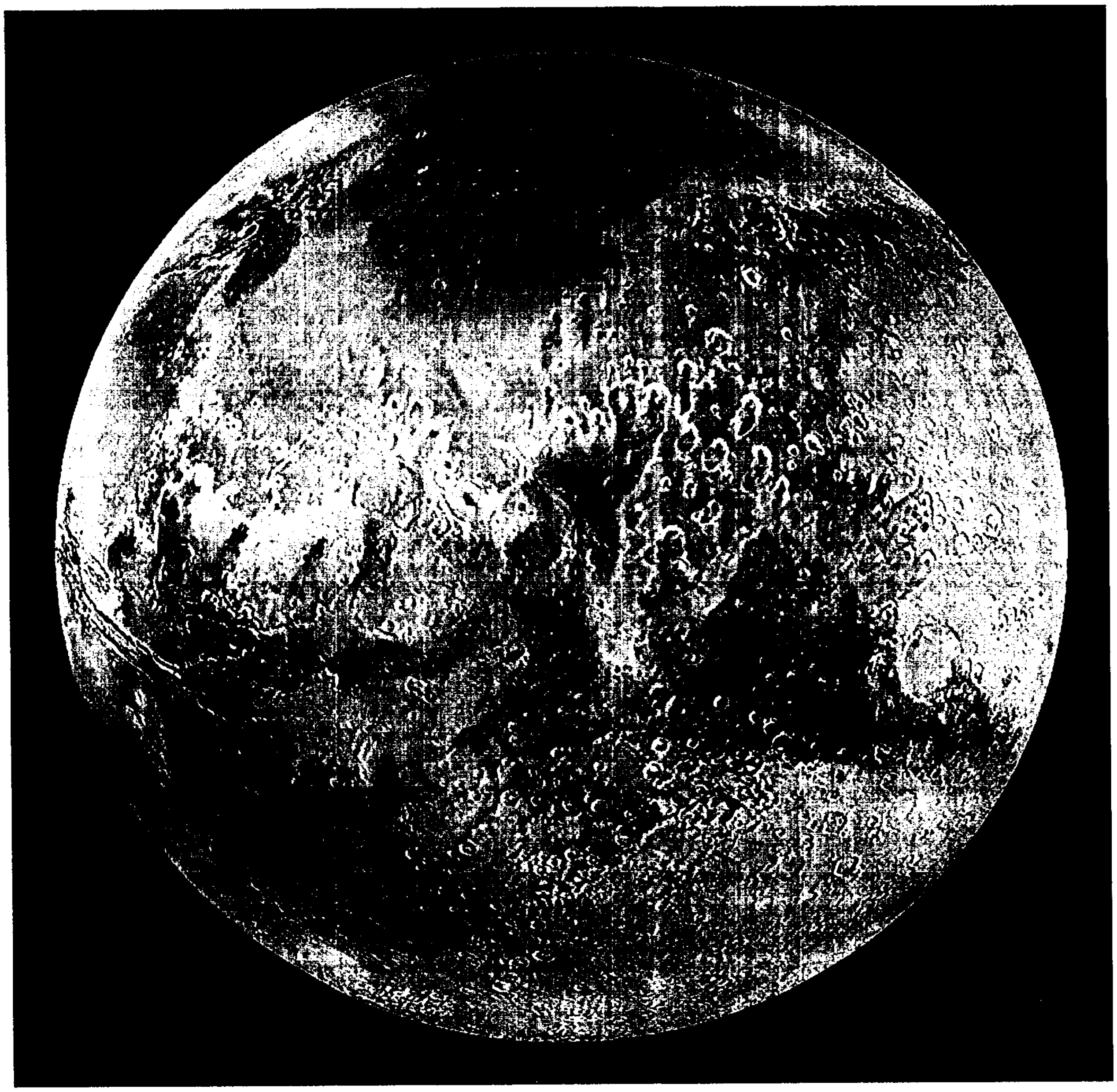


Figure 12. A global view of Mars centered at the Ares Vallis and Tiu Vallis regions

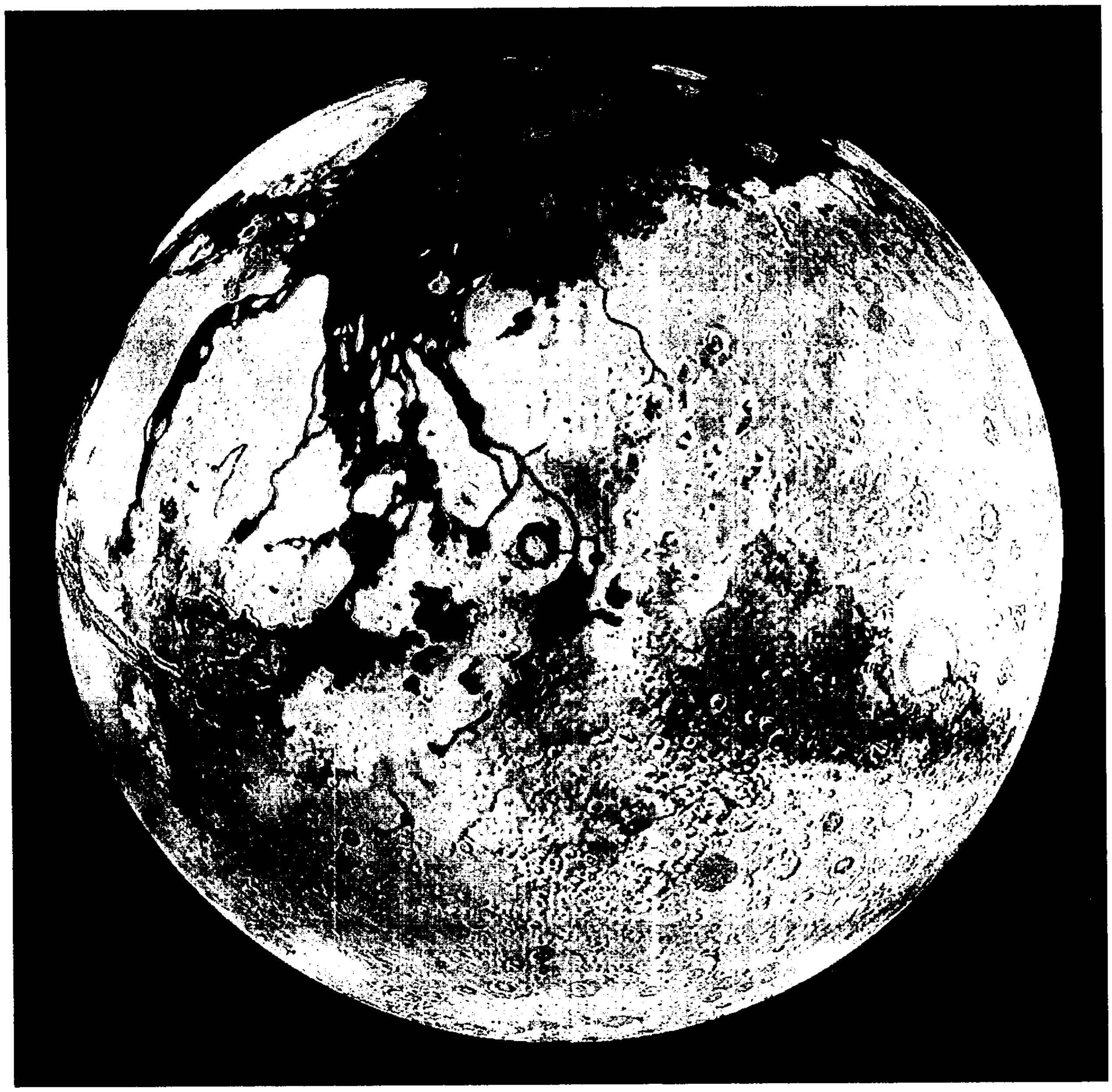


Figure 13. A golbal view of Mars with an overlay of the Mars geologic map.

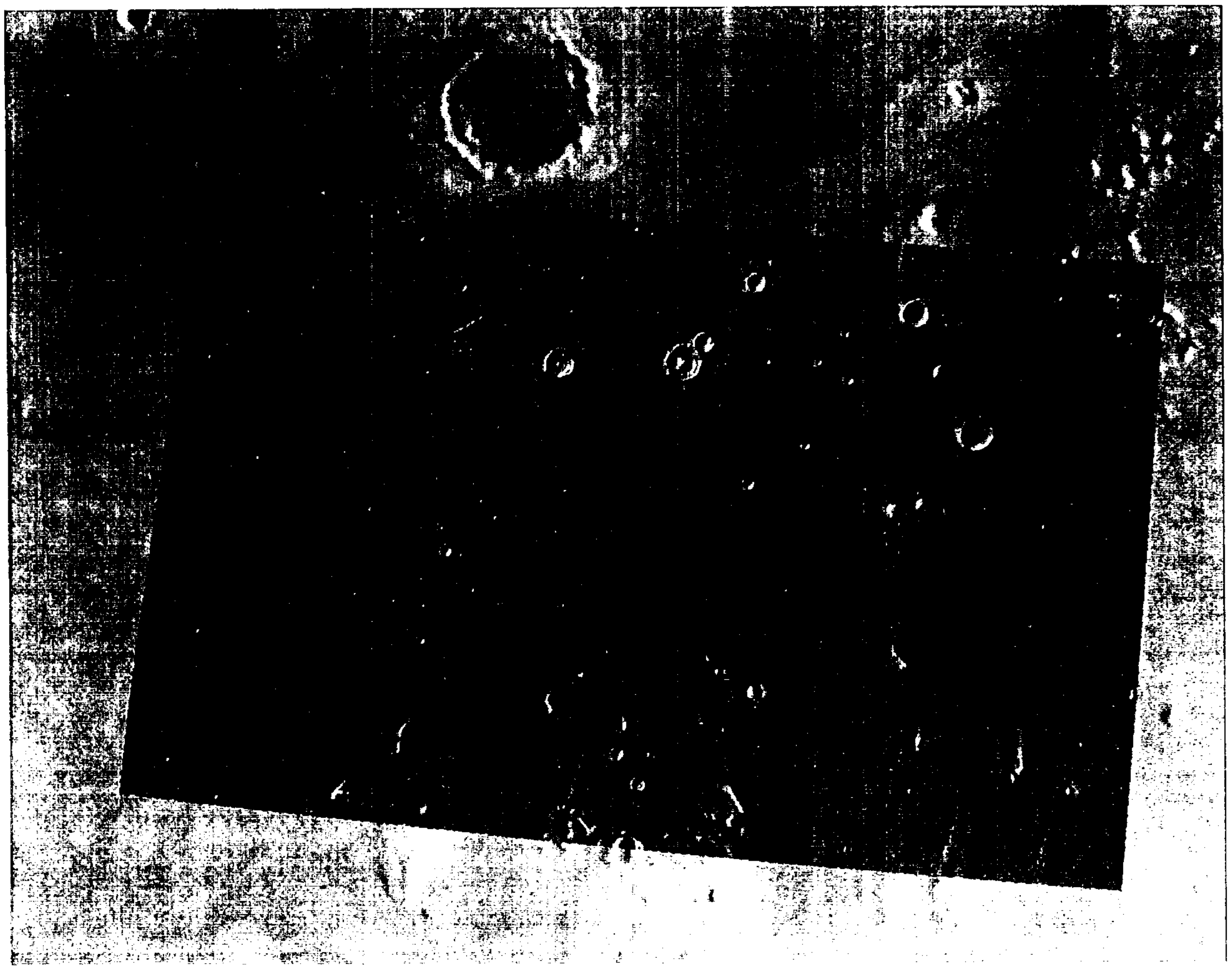


Figure 14. Ares Vallis of Mars, rendered with a 30 meter resolution patch on top of the Mars global color mosaic.